

5 Appendix A: exemplary source code for thor32.dll.

```
//-----
#include <windows.h>
#include <TriceratMessaging.h>
HINSTANCE InstanceHandle;
10 bool Processed = false;
extern "C" __declspec(dllexport) void LoadThorA();
extern "C" __declspec(dllexport) void UnloadThorA();
// Shared Data
# pragma data_seg(".shared") // Make a new section that we'll make shared
15 HHOOK hHook = 0;           // HHOOK from SetWindowsHook
# pragma data_seg()
LRESULT CALLBACK GetMsgProc(int code, WPARAM wParam, LPARAM lParam);
# pragma argsused
int WINAPI DllEntryPoint(HINSTANCE hinst, unsigned long reason, void* lpReserved)
20 {
    HWND hWndMjolnir = NULL;
    InstanceHandle = hinst;
    DisableThreadLibraryCalls(hinst);
    if (!Processed)
25    {
        Processed = true;
        hWndMjolnir = FindWindow("TMjolnirMainForm", NULL);
        if (NULL != hWndMjolnir)
        {
30            SendMessage(hWndMjolnir, TM_D2K_CHECKALLOWEDAPP, 0,
                         GetCurrentProcessId());
        }
        return true;
    }
35 //-----
void LoadThorA()
{
    hHook = SetWindowsHookEx(WH_CALLWNDPROC, (HOOKPROC)GetMsgProc, InstanceHandle, 0 );
}
40 //-----
void UnloadThorA()
{
    UnhookWindowsHookEx(hHook);
    hHook = NULL;
45 }
//-----
LRESULT CALLBACK GetMsgProc(int code, WPARAM wParam, LPARAM lParam)
{
    LRESULT retValue = 0;
50    HWND hWndMjolnir = NULL;
    if (!Processed)
    {
        Processed = true;
        hWndMjolnir = FindWindow("TMjolnirMainForm", NULL);
55        if (NULL != hWndMjolnir)
        {
            SendMessage(hWndMjolnir, TM_D2K_CHECKALLOWEDAPP, 0,
                         GetCurrentProcessId());
        }
    }
    retValue = CallNextHookEx(hHook, code, wParam, lParam);
60    return retValue;
} //-----
```

5 Appendix B: exemplary source code for mjolnir.exe.

```
//-----  
#include <vcl.h>  
#pragma hdrstop  
#include "MjolnirMainUnit.h"  
#include "UnallowedAppUnit.h"  
//-----  
#pragma package(smart_init)  
#pragma link "NetworkInfo"  
#pragma link "StBase"  
15 #pragma link "StShBase"  
#pragma link "StTrIcon"  
#pragma link "NetworkInfo"  
#pragma resource "*.*.dfm"  
#pragma link "psapi.lib"  
20 typedef __stdcall bool (*LOADDLL)();  
typedef __stdcall bool (*UNLOADDLL)();  
static bool KillUserProcess(DWORD ProcessId);  
TMjolnirMainForm *MjolnirMainForm;  
//-----  
25 __fastcall TMjolnirMainForm::TMjolnirMainForm(TComponent* Owner)  
    : TForm(Owner)  
{  
    PmpStarting = false;  
    DesktopStarting = false;  
30    SwingMjolnir = false;  
}  
//-----  
void __fastcall TMjolnirMainForm::HookThor32()  
{  
    35    if (NULL == hThor32Lib)  
    {  
        ShowMessage("Unable to load Thor32.Dll");  
        Close();  
    }  
    40    LOADDLL pfnLoadDll = (LOADDLL)GetProcAddress(hThor32Lib,  
        "LoadThorA");  
    (*pfnLoadDll)();  
}  
void __fastcall TMjolnirMainForm::UnhookThor32()  
45 {  
    if (NULL == hThor32Lib)  
    {  
        ShowMessage("Unable to load Thor32.Dll");  
        Close();  
    }  
    50    UNLOADDLL pfnUnloadDll = (UNLOADDLL)GetProcAddress(hThor32Lib,  
        "UnloadThorA");  
    (*pfnUnloadDll)();  
}  
void __fastcall TMjolnirMainForm::FormCreate(TObject *Sender)  
55 {  
    TRegistry *Reg = new TRegistry();  
    bool ThorIsDisabled = false;  
    ShowWindow(Application->Handle, SW_HIDE);  
    Session->Active = false;  
60    ProductID = TI_PRODUCT_DESK2K1;
```

```
5     Application->CreateForm(__classid(TLicenseForm), &LicenseForm);
if (!LicenseForm->ValidateLicense())
{
    MessageBox(NULL, "desktop 2001 License has expired!", "triCerat License",
10      MB_OK | MB_ICONERROR | MB_SYSTEMMODAL);
Application->Terminate();
return;
}
delete LicenseForm;
Reg->RootKey = HKEY_LOCAL_MACHINE;
15 Reg->OpenKey("Software\\Tricerat\\Controls", true);
try
{
    ThorIsDisabled = Reg->ReadBool("DisableThor");
if (ThorIsDisabled)
20 {
    Reg->CloseKey();
    delete Reg;
    Application->Terminate();
    return;
}
}
catch(...)
{
}
30 Reg->CloseKey();
Reg->OpenKey("Software\\Tricerat\\Desktop 2001", true);
try
{
    LoadTimer->Interval = Reg->ReadInteger("MjolnirStartupDelay") * 1000;
350 if (0 >= LoadTimer->Interval)
{
    LoadTimer->Interval = 10000;
}
}
catch(...)
{
    Reg->WriteInteger("MjolnirStartupDelay", 10);
    LoadTimer->Interval = 10000;
}
40 Reg->CloseKey();
delete Reg;
TSecurity *sec = new TSecurity();
IsAdmin = sec->IsUserAdmin(getenv("COMPUTERNAME"), getenv("USERDOMAIN"),
getenv("USERNAME"));
45 delete sec;
hThor32Lib = LoadLibrary("Thor32.Dll");
wts = new TWtsTools();
HookThor32();
50 }
//-----
55 void __fastcall TMjolnirMainForm::FormActivate(TObject *Sender)
{
    ShowWindow(Application->Handle, SW_HIDE);
}
//-----
60 void __fastcall TMjolnirMainForm::FormClose(TObject *Sender, TCloseAction &Action)
{
```

```
5      UnhookThor32();
10     if (NULL != hThor32Lib)
11     {
12         FreeLibrary(hThor32Lib);
13     }
14     delete wts;
15 }
//-----
16 void __fastcall TMjolnirMainForm::FormHide(TObject *Sender)
17 {
18     ShowWindow(Application->Handle, SW_HIDE);
19     Top = 5000;
20     Left = 5000;
21 }
//-----
22 void __fastcall TMjolnirMainForm::HideTimerTimer(TObject *Sender)
23 {
24     Hide();
25 }
//-----
26 void __fastcall TMjolnirMainForm::HookBitBtnClick(TObject *Sender)
27 {
28     HookThor32();
29 }
//-----
30 void __fastcall TMjolnirMainForm::UnhookBitBtnClick(TObject *Sender)
31 {
32     UnhookThor32();
33 }
//-----
34 void __fastcall TMjolnirMainForm::AddOwners(TStrings* sql)
35 {
36     sql->Add(" IN (SELECT ID FROM Owners WHERE Name = '" +
37             FNetworkInfo->UserName + "'");
38     if (FNetworkInfo->LocalComputerName != ("\\\" + FNetworkInfo->DomainName))
39     {
40         FNetworkInfo->SourceServerName = FNetworkInfo->DomainControllerName;
41         for (int i = 0; i < FNetworkInfo->MyGlobalGroupCount; i++)
42             sql->Add(" OR Name = '" + FNetworkInfo->MyGlobalGroupNames[i] + "'");
43     }
44     FNetworkInfo->SourceServerName = "";
45     for (int i = 0; i < FNetworkInfo->MyLocalGroupCount; i++)
46         sql->Add(" OR Name = '" + FNetworkInfo->MyLocalGroupNames[i] + "'");
47     if (!FNetworkInfo->ClientName.IsEmpty())
48         sql->Add(" OR Name = '" + FNetworkInfo->ClientName + "'");
49     if (!FNetworkInfo->LocalComputerName.IsEmpty())
50         sql->Add(" OR Name = '" + FNetworkInfo->LocalComputerName + "'");
51     sql->Add(")");
52 }
//-----
53 void __fastcall TMjolnirMainForm::StringGridInitialize()
54 {
55     AllowedAppsStringGrid->RowCount = 1;
56     AllowedAppsStringGrid->FixedRows = 0;
57     AllowedAppsStringGrid->ColCount = 3;
58     AllowedAppsStringGrid->ColWidths[0] = 100;
59     AllowedAppsStringGrid->ColWidths[1] = 400;
60     AllowedAppsStringGrid->ColWidths[2] = 50;
```

```
5     AllowedAppsStringGrid->Refresh();
FirstRowOfStringGrid = true;
}
void __fastcall TMjolnirMainForm::LoadAllowedExecutables()
{
10    Session->Active = true;
TQuery* query = new TQuery(NULL);
AnsiString ParentProcess;
AnsiString ProcessName;
char szFileShortPath[MAX_PATH] = "unknown";
int InstanceLimit = 0;
int j = 0;
StringGridInitialize();
query->UniDirectional = true;
query->Constrained = true;
query->RequestLive = false;
query->DatabaseName = "Tricerat D2K1";
query->SQL->Add("SELECT DISTINCT e.Executable, e.InstanceLimit, e.Dependencies ");
query->SQL->Add("FROM Executables e, StartMenuItems s, Owners o ");
query->SQL->Add("WHERE ");
25    query->SQL->Add("e.ID = s.ExecutableID AND s.OwnerID = o.ID ");
query->SQL->Add("AND e.Disabled = False ");
query->SQL->Add("AND s.OwnerID ");
AddOwners(query->SQL);
try
30    {
        query->Open();
        for (int i = 0; i < query->RecordCount; i++)
        {
            try
            {
                InstanceLimit = query->FieldByName("InstanceLimit")->AsInteger;
            }
            catch(...)
            {
                InstanceLimit = 1;
            }
            ProcessName = query->FieldByName("Executable")->AsString;
            if (0 != ExtractFileExt(ProcessName).AnsiCompareIC(".EXE"))
40            {
                ProcessName = GetFileAssociation(ProcessName);
            }
            ParentProcess =
                ExtractFileName(ProcessName);
            AddAllowedApp(ParentProcess, ProcessName, InstanceLimit);
            AddDependencies(query->FieldByName("Dependencies")->AsString,
                ParentProcess, 9999);
            query->Next();
        }
    }
55    catch (...)

    {
    }
    query->Close();
    query->SQL->Clear();
    query->SQL->Add("SELECT DISTINCT e.Executable, e.InstanceLimit, e.Dependencies ");
    query->SQL->Add("FROM Executables e, DesktopItems d, Owners o ");
    query->SQL->Add("WHERE ");
60
```

```

5      query->SQL->Add("e.ID = d.ExecutableID AND d.OwnerID = o.ID ");
6      query->SQL->Add("AND e.Disabled = False ");
7      query->SQL->Add("AND d.OwnerID ");
8      AddOwners(query->SQL);
9      try
10     {
11         query->Open();
12         for (int i = 0; i < query->RecordCount; i++)
13         {
14             try
15             {
16                 InstanceLimit = query->FieldByName("InstanceLimit")->AsInteger;
17             }
18             catch(...)
19             {
20                 InstanceLimit = 1;
21             }
22             ProcessName = query->FieldByName("Executable")->AsString;
23
24             if (0 != ExtractFileExt(ProcessName).AnsiCompareIC(".EXE"))
25             {
26                 ProcessName = GetFileAssociation(ProcessName);
27             }
28             ParentProcess =
29                 ExtractFileName(ProcessName);
30             AddAllowedApp(ParentProcess, ProcessName, InstanceLimit);
31             AddDependencies(query->FieldByName("Dependencies")->AsString,
32                             ParentProcess, 9999);
33             query->Next();
34         }
35     }
36     catch (...)
37     {
38     }
39     query->Close();
40     delete query;
41     SwingMjolnir = true;
42     Session->Active = false;
43 }
44 //-----
45 bool __fastcall TMjolnirMainForm::AddAllowedApp(AnsiString ParentProcess, AnsiString AppPath, int Instances)
46 {
47     //The TStringGrid has one row initialy, so don't add a new one.
48     if (FirstRowOfStringGrid)
49     {
50         FirstRowOfStringGrid = false;
51     }
52     else
53     {
54         AllowedAppsStringGrid->RowCount++;
55     }
56     AllowedAppsStringGrid->Cells[0][AllowedAppsStringGrid->RowCount - 1] =
57         ParentProcess;
58     AllowedAppsStringGrid->Cells[1][AllowedAppsStringGrid->RowCount - 1] =
59         ResolveFileShortPath(AppPath);
60     AllowedAppsStringGrid->Cells[2][AllowedAppsStringGrid->RowCount - 1] =
61         String(Instances);
62     return true;

```

```
5     }
void __fastcall TMjolnirMainForm::OnCheckAllowedApp(TMessage &Message)
{
    ValidateProcess(Message.LParam);
}
10 //-----
bool __fastcall TMjolnirMainForm::ValidateProcess(DWORD ProcessId)
{
    AnsiString ProcessPath;
    TStringList *RunningApps;
15    bool InstanceCountExceeded = false;
    bool ParentProcessRunning = false;
    bool ValidProcess = false;
    int i = 0;
    if (! SwingMjolnir)
20    {
        return true;
    }
    ProcessPath = GetProcessShortPath(ProcessId);
if (ProcessPath.IsEmpty())
25    {
        return true;
    }
//Get the list of running apps.
RunningApps = wts->GetSessionProcessList();
30 //Go through the Allowed Apps Grid and see if the Process is allowed to run.
i = -1;
while (AllowedAppsStringGrid->RowCount > ++i)
{
    if (0 == ProcessPath.AnsiCompareIC(
35       AllowedAppsStringGrid->Cells[1][i]))
    {
        int AppCount = 0;
        int j = 0;

40        //Check the instance count
        j = -1;
        while(RunningApps->Count > ++j)
        {
            if (0 == RunningApps->Strings[j].AnsiCompareIC(
45               ExtractFileName(AllowedAppsStringGrid->Cells[1][i])))
            {
                AppCount++;
            }
        }
50        if (AppCount > AllowedAppsStringGrid->Cells[2][i].ToIntDef(0))
        {
            InstanceCountExceeded = true;
        }
55        //Try to find the Parent process.
        j = -1;
        while(RunningApps->Count > ++j)
        {
            if (0 == RunningApps->Strings[j].AnsiCompareIC(
60               AllowedAppsStringGrid->Cells[0][i]))
            {
                ParentProcessRunning = true;
            }
        }
    }
}
```

```

5         }
6     }
7     if (!InstanceCountExceeded && ParentProcessRunning)
8     {
9         ValidProcess = true;
10        break;
11    }
12}
13RunningApps->Clear();
14delete RunningApps;
15//Validate the Instance Count.
16if (InstanceCountExceeded)
17{
18    if (IsAdmin)
19    {
20        MessageBox(NULL, "The program Instance Count has been exceeded.\n\nPlease adjust the program \"Instance Count Limit\".", "Instance CCount Exceeded", MB_OK | MB_SYSTEMMODAL | MB_ICONINFORMATION);
21    }
22    else
23    {
24        if (KillUserProcess(ProcessId))
25        {
26            TInstanceLimitForm *Notify = new TInstanceLimitForm(NULL);
27            Notify->ProcessPathLabel->Caption = ProcessPath;
28            Notify->ShowModal();
29            delete Notify;
30        }
31    }
32    return false;
33}
34//Validate the process.
35if (!ValidProcess)
36{
37    if (IsAdmin)
38    {
39        TAdminForm *admin = new TAdminForm(NULL);
40        admin->ProcessEdit->Text = ProcessPath;
41        admin->ShowModal();
42        delete admin;
43    }
44    else
45    {
46        if (KillUserProcess(ProcessId))
47        {
48            TUnallowedAppForm *Notify = new TUnallowedAppForm(NULL);
49            Notify->ProcessPathLabel->Caption = ProcessPath;
50            Notify->ShowModal();
51            delete Notify;
52        }
53    }
54    return false;
55}
56return true;
57}
58
59
60AnsiString __fastcall TMjolnirMainForm::ResolveFileShortPath(AnsiString File)
61{

```

```

5     AnsiString Path;
6     AnsiString FileShortPath;
7     char szFileShortPath[MAX_PATH] = "unknown";
8     TDirTools *Dir = new TDirTools();
9     File = Dir->ParseEnvironment(File);
10    delete Dir;
11    Path = getenv("PATH");
12    Path = ".\\;" + Path;
13    if (ExtractFilePath(File).IsEmpty())
14    {
15        //For some reason, FileSearch() does not search the CurrentDir.
16        if (FileExists(GetCurrentDir() + "\\\" + File))
17        {
18            File = GetCurrentDir() + "\\\" + File;
19        }
20    }
21    else
22    {
23        File = FileSearch(File, Path);
24    }
25    GetShortPathName(File.c_str(), szFileShortPath,
26                      sizeof(szFileShortPath));
27    FileShortPath = szFileShortPath;
28    if (FileShortPath.IsEmpty())
29    {
30        FileShortPath = File;
31    }
32    return FileShortPath;
33}
34//-----
35AnsiString __fastcall TMjolnir MainForm::GetProcessShortPath(DWORD ProcessId)
36{
37    HANDLE hProcess;
38    HMODULE hMod;
39    DWORD cbNeeded = 0;
40    char szProcessPath[MAX_PATH] = "unknown";
41    char szProcessShortPath[MAX_PATH] = "unknown";
42    AnsiString ProcessShortPath;
43    hProcess = OpenProcess(PROCESS_QUERY_INFORMATION | PROCESS_VM_READ, FALSE,
44                          ProcessId);
45    if (EnumProcessModules(hProcess, &hMod, sizeof(hMod), &cbNeeded))
46    {
47        //To get just the name of the process, call this:
48        //GetModuleBaseName(hProcess, hMod, szProcessName, sizeof(szProcessName));
49        //Get the full path of the process.
50        GetModuleFileNameEx(hProcess, hMod, szProcessPath, sizeof(szProcessPath));
51        GetShortPathName(szProcessPath, szProcessShortPath,
52                        sizeof(szProcessShortPath));
53        ProcessShortPath = szProcessShortPath;
54    }
55    CloseHandle(hProcess);
56    return ProcessShortPath;
57}
58//-----
59bool KillUserProcess(DWORD ProcessId)
60{
61    HANDLE hProcess;
62    hProcess = OpenProcess(PROCESS_ALL_ACCESS, TRUE, ProcessId);

```

```
5     if (NULL == hProcess)
6         return false;
7     return TerminateProcess(hProcess, 0);
8 }
9 //-----
10 void __fastcall TMjolnirMainForm::OnDesktopInit(TMessage & Message)
11 {
12     if (0 == Message.WParam)
13     {
14         DesktopStarting = true;
15         LoadTimer->Enabled = false;
16     }
17     if (1 == Message.WParam)
18     {
19         DesktopStarting = false;
20         if (!PmpStarting)
21         {
22             InitializeMjolnir();
23         }
24     }
25 }
26 void __fastcall TMjolnirMainForm::OnPmpInit(TMessage & Message)
27 {
28     if (0 == Message.WParam)
29     {
30         PmpStarting = true;
31         LoadTimer->Enabled = false;
32     }
33     if (1 == Message.WParam)
34     {
35         PmpStarting = false;
36         if (!DesktopStarting)
37         {
38             InitializeMjolnir();
39         }
40     }
41 }
42 void __fastcall TMjolnirMainForm::InitializeMjolnir()
43 {
44     LoadAllowedExecutables();
45 }
46 void __fastcall TMjolnirMainForm::OnRefresh(TMessage & Message)
47 {
48     InitializeMjolnir();
49 }
50 void __fastcall TMjolnirMainForm::LoadTimerTimer(TObject *Sender)
51 {
52     LoadTimer->Enabled = false;
53     InitializeMjolnir();
54 }
55 //-----
56 void __fastcall TMjolnirMainForm::RefreshBitBtnClick(TObject *Sender)
57 {
58     InitializeMjolnir();
59 }
60 //-----
61 void __fastcall TMjolnirMainForm::AddDependencies(AnsiString Delimited,
62     AnsiString ParentProcess, int DefaultInstanceLimit)
```

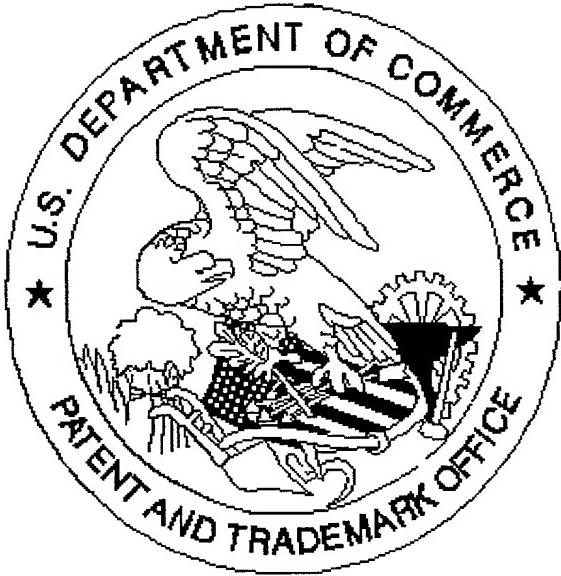
```

5   {
6     TStringList *ParsedStrings;
7     int i = 0;
8     if (Delimited.IsEmpty())
9       return;
10    ParsedStrings = GetParsedStringList(Delimited);
11    i = -1;
12    while (ParsedStrings->Count > ++i)
13    {
14      AddAliasDependencies(ParsedStrings->Strings[i],
15        ParentProcess, DefaultInstanceLimit);
16    }
17  }
18  TStringList * __fastcall TMjolnirMainForm::GetParsedStringList(AnsiString Delimited)
19  {
20    TStringList *DelimitedCharList = new TStringList;
21    TStringList *ParsedStrings = new TStringList();
22    TStringList *SubStrings;
23    AnsiString FoundString;
24    AnsiString RemainingString;
25    bool SubStringsFound = false;
26    int Index = 0;
27    int i = 0;
28    int j = 0;
29    if (Delimited.IsEmpty())
30      return ParsedStrings;
31    DelimitedCharList->Add(";");
32    DelimitedCharList->Add(",");
33    i = -1;
34    while (DelimitedCharList->Count > ++i)
35    {
36      Index = Delimited.AnsiPos(DelimitedCharList->Strings[i]);
37
38      if (0 >= Index)
39        continue;
40      SubStringsFound = true;
41      FoundString = Delimited.SubString(1, Index - 1);
42      RemainingString = Delimited.SubString(Index + 1,
43        Delimited.Length() - Index);
44      SubStrings = GetParsedStringList(FoundString);
45      j = -1;
46      while (SubStrings->Count > ++j)
47      {
48        ParsedStrings->Add(SubStrings->Strings[j]);
49      }
50      delete SubStrings;
51      SubStrings = GetParsedStringList(RemainingString);
52      j = -1;
53      while (SubStrings->Count > ++j)
54      {
55        ParsedStrings->Add(SubStrings->Strings[j]);
56      }
57      delete SubStrings;
58    }
59    if (!SubStringsFound)
60    {
61      ParsedStrings->Add(Delimited);
62    }

```

```
5     DelimitedCharList->Clear();
       delete DelimitedCharList;
       return ParsedStrings;
    }
void __fastcall TMjolnirMainForm::AddAliasDependencies(AnsiString Alias,
10    AnsiString ParentProcess, int DefaultInstanceLimit)
{
    if (FileExists(ResolveFileShortPath(Alias)))
    {
        AddAllowedApp(ParentProcess, Alias, DefaultInstanceLimit);
15    return;
    }
    TQuery* query = new TQuery(NULL);
    query->UniDirectional = true;
    query->Constrained = true;
20    query->RequestLive = false;
    query->DatabaseName = "Tricerat D2K1";
    query->SQL->Add("SELECT d.Path ");
    query->SQL->Add("FROM Dependencies d ");
    query->SQL->Add("WHERE ");
25    query->SQL->Add("d.Name = '" + Alias + "'");
    try
    {
        query->Open();
        for (int i = 0; i < query->RecordCount; i++)
30    {
        AddAllowedApp(ParentProcess,
                      ResolveFileShortPath(query->FieldByName("Path")->AsString),
                      DefaultInstanceLimit);
        query->Next();
35    }
    }
    catch (...)
    {
    }
40    query->Close();
    delete query;
}
AnsiString __fastcall TMjolnirMainForm::GetFileAssociation(AnsiString File)
{
45    AnsiString Association;
    AnsiString FilePath;
    AnsiString FileName;
    char szResult[1024];
    ZeroMemory(szResult, sizeof(szResult));
50    Association = File;
    File = ResolveFileShortPath(File);
    if (FileExists(File))
    {
55        FileName = ExtractFileName(File);
        FilePath = ExtractFilePath(File);
        if (!FileName.IsEmpty() && !FilePath.IsEmpty())
        {
            if (32 < (int)FindExecutable(FileName.c_str(), FilePath.c_str(), szResult))
            {
60            if (FileExists(szResult))
            {
                Association = szResult; } } } return Association;}
```

United States Patent & Trademark Office
Office of Initial Patent Examination -- Scanning Division



Application deficiencies found during scanning:

Page(s) _____ of _____ were not present
for scanning. (Document title)

Page(s) _____ of _____ were not present
for scanning. (Document title)

Scanned copy is best available. FIG-1 IS DARK